

Sistemas Operacionais

MAC 422

EP 3

Julia Leite
Eduardo Brancher

Detalhes de implementação





Arquivos do EP

- `main.cpp`
 - implementação do simulador de sistema de arquivos
- `definitions.hpp`
 - header com as definições, como o símbolo usado como separador de bloco
- `aux.hpp`
 - header com funções auxiliares
- `classes.hpp`
 - header com as estruturas de dados utilizadas
- `Makefile`



Definições

definitions.hpp

➤ Adotamos :

- Tamanho do bloco no sistema de arquivos
 - `BLOCK_SIZE = 4 kB = 4096 bytes`
- Tamanho máximo do sistema de arquivos
 - `MAX_SIZE = 100 MB`
- Número de blocos do sistema:
 - `N_BLOCKS = MAX_SIZE/BLOCK_SIZE = 24414`



Definições

definitions.hpp

- Adotamos :
 - Separador de blocos
 - `BLOCK_TERMINATOR = "_"`
 - Separador dos metadados de arquivos e diretórios:
 - `ARQ_SEPARATOR = "|"`
 - Indicador do final do arquivo
 - `TERMINATOR = "-1"`
- Observação : os blocos são separados pelo `BLOCK_TERMINATOR` entre espaços, com exceção dos blocos do bitmap, separados apenas pelo `BLOCK_TERMINATOR`, com a finalidade de poupar espaço



Definições

definitions.hpp

- Tamanho máximo do nome de diretórios e arquivos (incluindo a extensão): 30 caracteres
- Adotamos que um diretório pode ocupar apenas um bloco no sistema de arquivos, então em um diretório vazio há espaço para 33 diretórios ou 28 arquivos
- É possível que haja diretórios e arquivos em um diretório em qualquer combinação cuja soma de metadados não exceda 4096 bytes
 - Por exemplo, 15 arquivos e 15 diretórios seria válido



Arquivo que simula o sistema

*****.txt

- O sistema de arquivos é simulado em um .txt com 3 partes, respectivamente:
 - bitmap
 - FAT
 - / e demais diretórios e arquivos
- Os metadados do diretório / são armazenados logo depois da FAT, dentro do bloco onde ele armazena os metadados de seus filhos.
- <bitmap>_ FAT _ /(...)



Arquivo que simula o sistema

modelo ilustrativo

A seguir, um exemplo apenas ilustrativo com valores e tamanhos que não correspondem ao implementado

```
00000000000000001111111111_1111111111111111_<resto do bitmap>_  
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 14 -1 <resto da FAT> _  
|/|1607119474|1607119489|1607119489|12|  
|arq|frase.txt|1607001056|1607119480|1607119480|4379|13| _  
"O valor das coisas não pode ser medido pelo tempo que elas duram, mas pela  
intensidade com que elas acontecem. Por _ isso existem momentos inesquecíveis,  
coisas inexplicáveis e pessoas incomparáveis", Fernando Pessoa _  
_ _ <resto dos blocos> _ $-1$
```




Arquitetura do simulador

main.cpp

- Ao executar `mount arquivo.txt`, o programa processa e carrega o conteúdo do sistema de arquivos em uma estrutura de árvore com "/" como raiz, dessa forma, ele executa os comandos `cp`, `mkdir`, `rm`, etc sem consultar o arquivo `.txt`
 - não é possível executar comandos sem que um sistema de arquivos seja carregado (ou criado) pela `mount`
- Só é possível observar as modificações no arquivo `.txt` ao executar `umount`, que grava as alterações no `.txt`, completando a simulação do sistema



Observações

- Ocasionalmente, a função `rmdir` não funciona corretamente
 - Apesar de remover o diretório solicitado da árvore, em alguns casos, alguns subdiretórios não são removidos, apesar de não poderem ser referenciados, já que o “diretório pai” foi apagado
 - Então, as informações exibidas pelo `df ()` podem apresentar inconsistência, pois os dados desses sub-diretórios não removidos não serão apagados
- Essa inconsistência é solucionada ao executar `umount`, já que apenas os dados dos diretórios e arquivos presentes na árvore de diretórios são considerados para escrever o arquivo que simula o sistema
- Não foi possível consertar o problema devido ao prazo

Experimentos





Observações sobre os experimentos:

- Os experimentos foram realizados em uma máquina Intel core i7, 8th Generation
 - 8 cores
 - Ubuntu 18.04

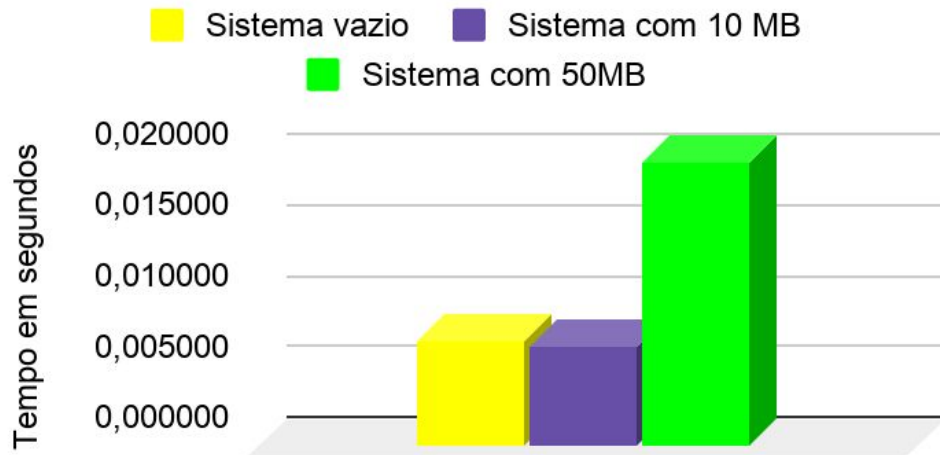


Experimentos

- Foram realizados os experimentos pedidos, para comparar o efeito que a ocupação de espaço do sistema de arquivos teria sobre sua eficiência.
- Por conta da arquitetura do programa, a ocupação de espaço não deve interferir demasiadamente no tempo de execução dos comandos, já que não manipulamos o arquivo `.txt`, a menos que `umount` seja executado
- Assumimos uma distribuição normal para o cálculo do intervalo de confiança

Cópia de arquivo de 1 MB

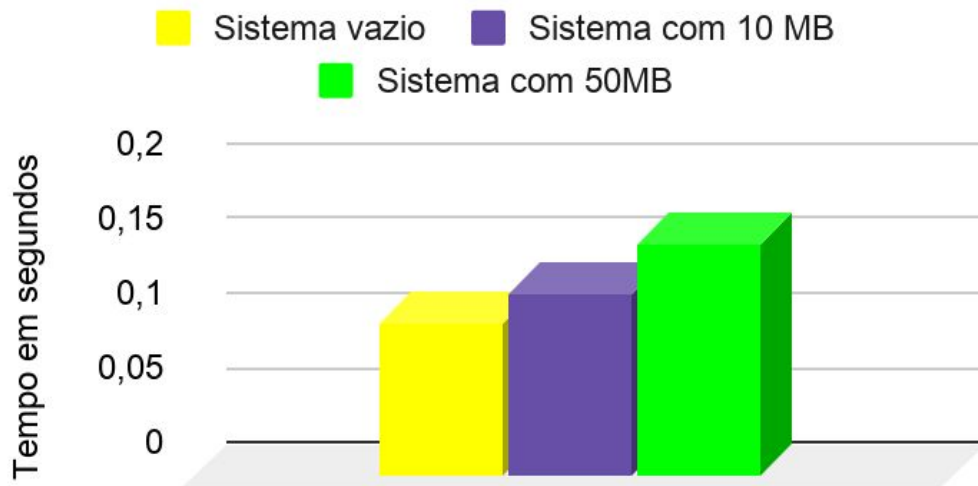
Cópia de um arquivo de 1MB



- sistema de arquivos vazio:
 - Média = 0,007374 s
 - Desvio = 0,001491 s
 - Intervalo de confiança = 0,000534
- Sistema com 10 MB ocupados:
 - Média = 0,006878 s
 - Desvio = 0,000931 s
 - Intervalo de confiança = 0,003995
- Sistema com 50 MB ocupados:
 - Média = 0,019963 s
 - Desvio = 0,002988 s
 - Intervalo de confiança = 0,001069

Cópia de arquivo de 10 MB

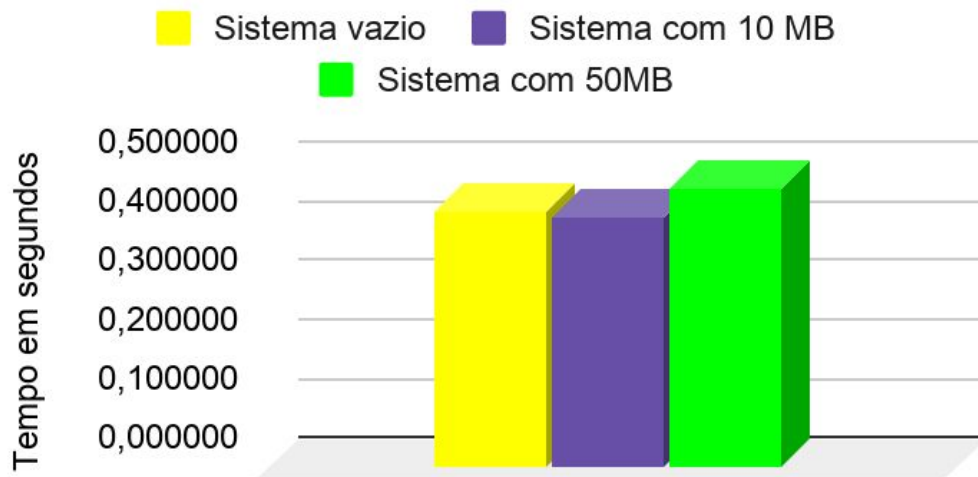
Cópia de um arquivo de 10 MB



- sistema de arquivos vazio:
 - Média = 0,101847 s
 - Desvio = 0,011164 s
 - Intervalo de confiança = 0,000333
- Sistema com 10 MB ocupados
 - Média = 0,121313 s
 - Desvio = 0,168895 s
 - Intervalo de confiança = 0,060437
- Sistema com 50 MB ocupados
 - Média = 0,153278 s
 - Desvio = 0,23764s
 - Intervalo de confiança = 0,008564

Cópia de arquivo de 30 MB

Cópia de arquivo de 30 MB



- sistema de arquivos vazio:
 - Média = 0,430606 s
 - Desvio = 0,021610 s
 - Intervalo de confiança = 0,007733
- Sistema com 10 MB ocupados
 - Média = 0,419305 s
 - Desvio = 0,022843 s
 - Intervalo de confiança = 0,008174
- Sistema com 50 MB ocupados
 - Média = 0,467998 s
 - Desvio = 0,262249 s
 - Intervalo de confiança = 0,093843

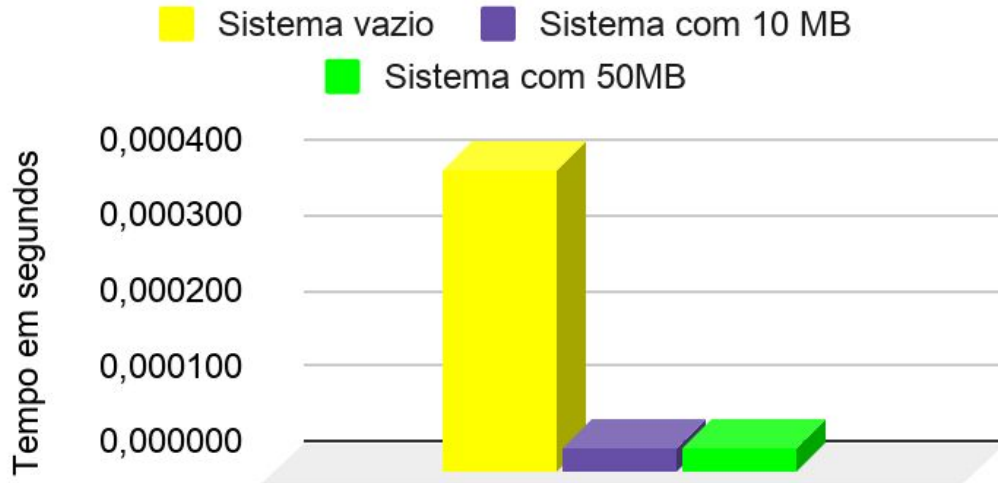


Discussão sobre os experimentos de cópia

- Nos experimentos, observamos pouca interferência da ocupação do sistema de arquivos no tempo de execução da cópia do arquivo, como esperado, já que não buscamos os dados do sistema no arquivo `.txt`, nem salvamos as alterações
 - Apenas alteramos o `bitmap` e `FAT` conforme necessário e adicionamos o arquivo na árvore
- Acreditamos que as variações observadas entre os experimentos com o mesmo tamanho de arquivo são fruto de variações aleatórias
- Observamos um incremento no tempo de execução conforme o tamanho do arquivo a ser copiado aumentava, comportamento esperado, por ser necessário manipular mais campos do `bitmap` e da `FAT`

Remoção de arquivo de 1 MB

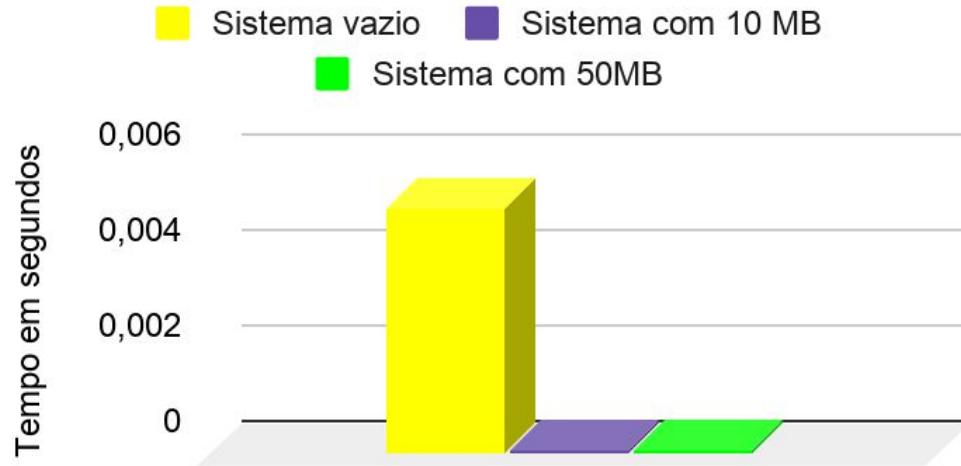
Remoção de arquivo de 1 MB



- sistema de arquivos vazio:
 - Média = 0,000396 s
 - Desvio = 0,000126 s
 - Intervalo de confiança = 0,000045
- Sistema com 10 MB ocupados
 - Média = 0,000031 s
 - Desvio = 0,000007s
 - Intervalo de confiança = 0,000003.
- Sistema com 50 MB ocupados
 - Média = 0,000031s
 - Desvio = 0,000009 s
 - Intervalo de confiança = 0,000003

Remoção de arquivo de 10 MB

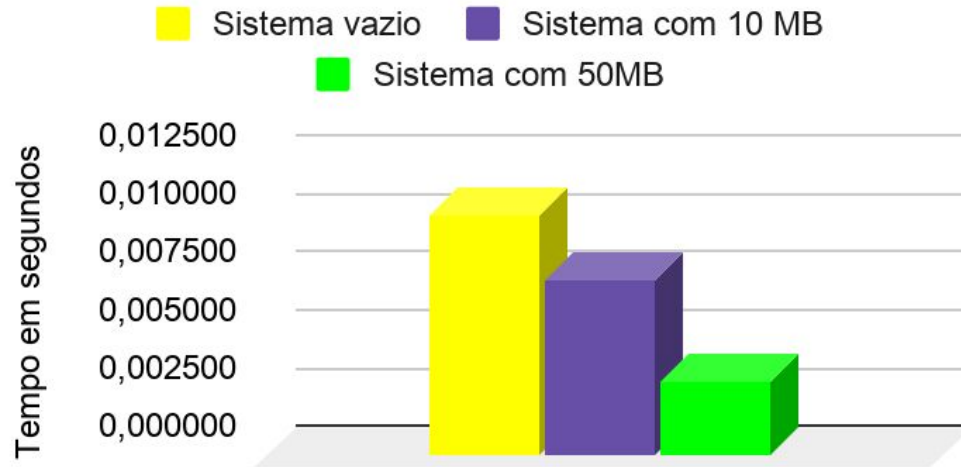
Remoção de arquivo de 10 MB




- sistema de arquivos vazio:
 - Média = 0,005089 s
 - Desvio = 0,001544 s
 - Intervalo de confiança = 0,000552
- Sistema com 10 MB ocupados
 - Média = 0,000052 s
 - Desvio = 0,000006 s
 - Intervalo de confiança = 0,000002.
- Sistema com 50 MB ocupados
 - Média = 0,000053s
 - Desvio = 0,000005 s
 - Intervalo de confiança = 0,000002

Remoção de arquivo de 30 MB

Remoção de arquivo de 30 MB



- sistema de arquivos vazio:
 - Média = 0,010186 s
 - Desvio = 0,002648 s
 - Intervalo de confiança = 0,000948
- Sistema com 10 MB ocupados
 - Média = 0,007515 s
 - Desvio = 0,001236s
 - Intervalo de confiança = 0,000442
- Sistema com 50 MB ocupados
 - Média = 0,003150s
 - Desvio = 0,003112 s
 - Intervalo de confiança = 0,001114

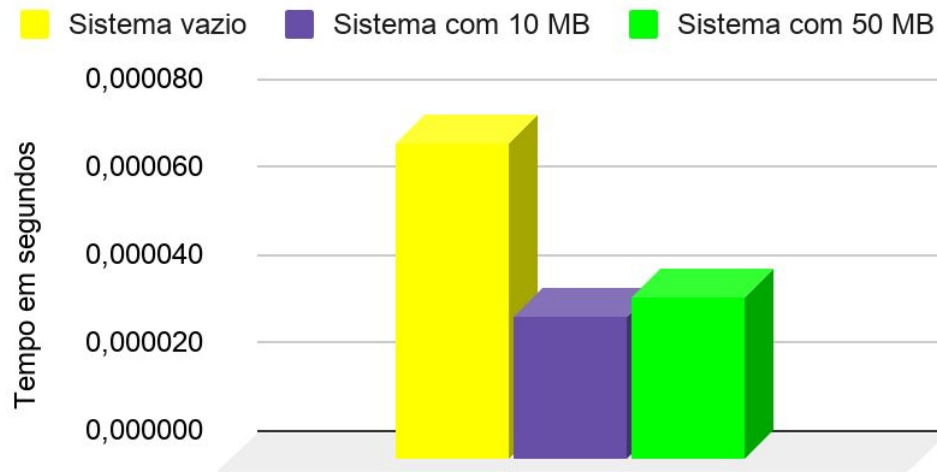


Discussão sobre os experimentos de remoção

- Observamos um aumento no tempo de execução conforme o tamanho do arquivo a ser removido crescia, o que é esperado, já que manipulamos mais campos do `bitmap` e da `FAT`
- Não sabemos, contudo, explicar o porquê do sistema de arquivos vazio ser o mais demorado, considerando o mesmo tamanho de arquivo
 - Talvez seja variação aleatória

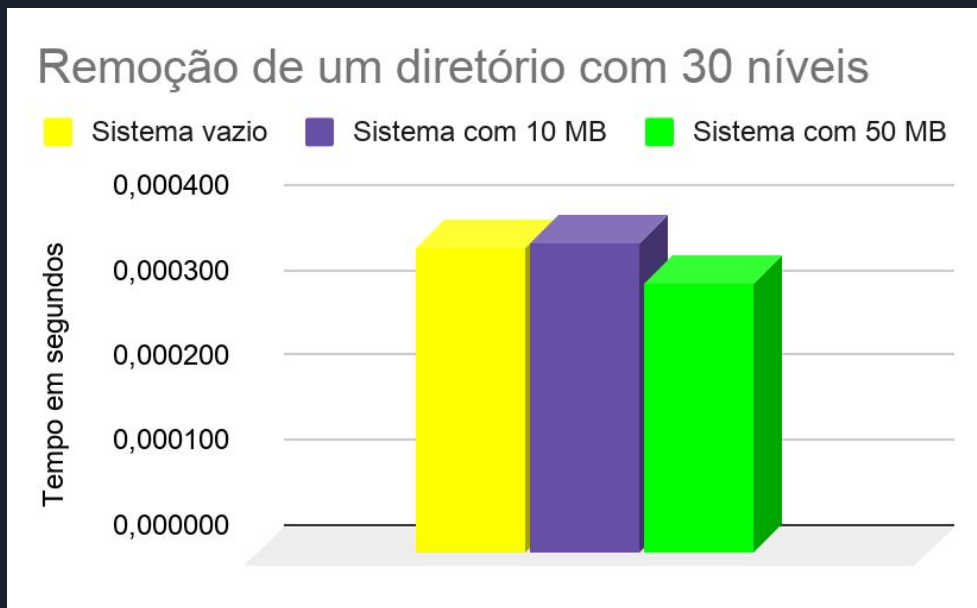
Remoção completa de um diretório com 30 níveis de hierarquia abaixo sem arquivos regulares nos subdiretórios

Remoção de um diretório com 30 níveis



- sistema de arquivos vazio:
 - Média = 0,000072 s
 - Desvio = 0,000049 s
 - Intervalo de confiança = 0,000017 s
- Sistema com 10 MB ocupados
 - Média = 0,000032 s
 - Desvio = 0,000004 s
 - Intervalo de confiança = 0,000001
- Sistema com 50 MB ocupados
 - Média = 0,000037 s
 - Desvio = 0,000009 s
 - Intervalo de confiança = 0,000003

Remoção completa de um diretório com 30 níveis de hierarquia abaixo com arquivos regulares nos subdiretórios (600 arquivos)



- sistema de arquivos vazio:
 - Média = 0,000360 s
 - Desvio = 0,000129s
 - Intervalo de confiança = 0,000046
- Sistema com 10 MB ocupados
 - Média = 0,000364 s
 - Desvio = 0,000116 s
 - Intervalo de confiança = 0,000041
- Sistema com 50 MB ocupados
 - Média = 0,000319s
 - Desvio = 0,000125s
 - Intervalo de confiança = 0,000045



Discussão dos Experimentos

- Os dois últimos experimentos correram como esperado
- A quantidade de espaço ocupado não deveria afetar a remoção de diretórios a partir do raiz, uma vez que se trata de uma busca em uma árvore
- O fato de haver muitos arquivos grandes no sistema de arquivos não afeta essa busca, de modo que a variação de tempo não apresenta correlação com a ocupação do sistema de arquivos, apenas com a quantidade de objetos a serem deletados



Participantes:

- 11221797: Julia Leite
- 8587409: Eduardo Brancher